Server

**Due Tuesday, July 28th, 11:59 PM**

This homework has the following learning objectives:

* Learn how to follow a network protocol (in this case, http)
* Learn network/socket programming
* Learn some simple multithreading

[**Github classroom invitation link**](https://www.google.com/url?q=https%3A%2F%2Fclassroom.github.com%2Fa%2FuuNTADEE&sa=D&sntz=1&usg=AFQjCNFjjB_IKpA-Fb-4cEjdD7KpQ0Ih3Q)

In this homework, we’ll look at the HTTP protocol as a Web server. After creating your personal repository, you will find that the hw5 directory contains:

* homework5.c: Skeleton code for the server side of a TCP application. This will be the primary file for this assignment, but feel free to modularize and create other files if you prefer to do so.
* WWW/: A directory containing example files for your Web server to distribute.
* thread\_example.c: Example code that illustrates a very simple threaded programming scenario. You are not required to use or make any changes to this file, but you should understand what it does.
* A Makefile. If you modularize your code into different files, make sure those changes are reflected in the Makefile (and don’t forget to git add those files to your personal git repo!).

Your server program will receive two arguments: 1) the port number it should listen on for incoming connections, and 2) the directory out of which it will serve files (often called the document root in production Web servers). For example:

$ ./homework5 8080 WWW

This command will tell your Web server to listen for connections on port 8080 and serve files out of the WWW directory. That is, the WWW directory is considered ‘/’ when responding to requests. For example, if you’re asked for /index.html, you should respond with the file that resides in WWW/index.html. If you’re asked for /dir1/dir2/file.ext, you should respond with the file WWW/dir1/dir2/file.ext.

**Requirements**

In addition to serving requested files, your server should handle at least the following cases:

1. HTML, text, and image files should all display properly. You’ll need to return the proper HTTP Content-Type header in your response, based on the file ending. Also, other binary file types like PDFs should be handled correctly. You don’t need to handle everything on that list, but you should at least be able to handle files with html, txt, jpeg, jpg, gif, png, and pdf extensions.
2. If asked for a file that does not exist, you should respond with a 404 error code (Not Found).
3. If asked for a directory, you should respond with a 400 code (Bad Request).
4. Some clients may be slow to complete a connection or send a request. Your server should be able to serve multiple clients concurrently, not just back-to-back. For this homework, use multithreading with pthreads to handle concurrent connections. We’ll try event-based concurrency in a future homework assignment.

When testing, you should be able to retrieve byte-for-byte copies of files from your server. Use wget or curl to fetch files and md5sum or diff to compare the fetched file with the original. We will do this when grading. *For full credit, the files need to be exact replicas of the original.*

**Tips**

* Please read the skeleton code to get a feel for what is available and what you should be doing.
* **As of the release of this assignment, we have not covered the concepts related to networking programming in class; to start on this homework, take a look at chapter 11 (especially 11.5 and 11.6) from the textbook to learn more about the network programming in C.**
* The labs onWednesday will help you get started on the assignment as well.
* Take compiler warnings seriously. Unless it’s an unused variable, you should address the warning as soon as you see it. Dealing with a pile of warnings just makes things more difficult later.
* Test your code in small increments. It’s much easier to localize a bug when you’ve only changed a few lines.
* If you need to copy a specific number of bytes from one buffer to another, and you’re not 100% sure that the data will be entirely text, use memcpy() rather than strncpy(). The latter terminates early if it finds a null terminator (‘\0’).
* If you’re trying to do some sort of specific string or memory manipulation, feel free to ask if there’s a better/recommended way to do it rather than brute force. Often there may be a standard library function that will make things easier.

**Roughly, your server should follow this sequence:**

1. Read the arguments, bind to the specified port, and find your document root (you might find the chdir() system call helpful).
2. Accept a connection, and hand it off to a new thread for concurrent processing.
3. Receive and parse a request from the client.
4. Look for the path that was requested, starting from your document root (the second argument to your program). One of three things should happen:
   1. If the path exists and it’s a file, formulate a response (with the Content-Type header set) and send it back to the client.
   2. If the path is a directory, respond with a 400 code (use stat() system call to examine a path).
   3. If the path does not exist, respond with a 404 code.
5. Close the connection, and continue serving other clients.

**Turn-in instructions and Grading**

You need to submit homework5.c to Gradescope. Grading will be done automatically using a script. We will publish this script after grading has completed; you are responsible for writing your own test cases. If you wish, you can share test cases you have written with the class.

**Reminders**

Always, always, always **check the return value of any system calls you make!** This is especially important for send, recv, read, and write calls that tell you how many bytes were read or written.

If you have any questions about the homework requirements or specification, please post on Piazza.